Differentiable collision detection for autonomous driving

Autonomous and Mobile Robotics
for MSc in Artificial Intelligence and Robotics
and MSc in Control Engineering
a.y. 2024-2025

Benedetta Rota
Id. *1936424 , MCER*
Camilla Rota
Id. *1936447 , MCER*
Simone Palumbo
Id. *1938214 , MARR*

Professor
Prof. Giuseppe Oriolo

Supervisor
Dott. Michele Cipriano

Contents

Abstract

Collision detection has a fundamental role in simulation, control, and learning for autonomous robotic systems. However, traditional methods for collision detection are inherently non-differentiable, which limits their integration with gradient-based optimization frameworks. To overcome this limitation, we have studied and implemented **DCOL** (Differentiable Collision Detection) [1], a fast and fully differentiable framework for handling collisions between a wide range of convex geometric primitives.

DCOL reformulates the collision detection problem as a convex optimization problem, computing the minimum uniform scaling factor required for the considered primitives to intersect. This differentiable property allows the smooth integration of DCOL with gradient-based optimization methods, facilitating the computation of smooth and collision-free trajectories in complex environments.

We have integrated DCOL into the **ALTRO** (Augmented Lagrangian Trajectory Optimization) framework, an advanced solver for trajectory optimization, demonstrating how collision constraints can be handled directly in the optimization process. The results highlight the effectiveness of DCOL in tackling complex problems such as the **"Piano Mover"** problem, the navigation of a **quadrotor** in a cluttered environment, and the passage of a **cone through an opening**. These examples demonstrate the flexibility of our approach and its potential impact on practical applications.
Our open-source Python implementation of DCOL is available at `https://github.com/DIAG-Robotics-Lab/AMR24-FP6-DCOL`.

1 Introduction

A key aspect in autonomous robotics is collision detection. The ability to determine the interaction between objects is crucial for ensuring the safety of people and environments, protecting the robot, and improving operational efficiency. However, classical methods for collision detection suffer from intrinsic limitations, in particular their non-differentiability and the inability to provide informative metrics in situations involving object penetration.
In this context, *Differentiable Collision Detection* (**DCOL**) represents a significant advancement.
The core of DCOL is the formulation of a convex optimization problem that determines the minimum uniform scaling necessary to cause an intersection between convex primitives [1]. This formulation allows to compute the scaling parameter in a differentiable way, providing a detailed collision metric even in the case of interpenetration. The ability to return such informative collision metric represents the biggest advantage over classical approaches.
In order to demonstrate the effectiveness and utility of this algorithm in motion planning and control problems, a trajectory optimization problem using DCOL as a *collision avoidance* framework has been implemented. The general optimization problem was implemented using the ALTRO solver [2], an algorithm that allows the management of constraints in solving an optimization problem. Specifically, three representative scenarios were analyzed to validate the approach: the "Piano Mover" problem, the passage of a cone through a narrow opening and the movement of a quadrotor in an environment densely populated with obstacles. These examples demonstrate the flexibility and effectiveness of DCOL in practical applications.

In this work, the topics introduced above will be presented in the following way:

- In Chap.2 some preliminary concepts necessary for understanding the problem are presented.
- In Chap.3 the **DCOL** is introduced with particular attention to the supported geometric primitives.
- In Chap.4 the **ALTRO** algorithm is presented as an algorithm for solving constrained optimization problems.
- In Chap.5 three scenarios are presented and analyzed: "Piano Mover", "Quadrotor" and "Cone Through an Opening".
- In Chap.6 interesting future applications are proposed.

2 Preliminary support

The aim of this chapter is to introduce the fundamental concepts that will serve as the theoretical foundation for the development of the work presented in the following chapters.

2.1 Collision detection in literature

This section provides an overview of the state of the art in collision detection algorithms, analyzing their main methods, evolutions and limitations.
Among the most well-known classical methods we find the **GJK** (Gilbert-Johnson-Keerthi) algorithm [3], widely used to determine the distance between convex objects. This algorithm uses the geometric representation through support points to identify whether two objects interact. Although GJK is very efficient in computational terms, it has some limitations, including its inability to provide detailed information in case of interpenetration between objects.

An evolution of GJK is its variant **Enhanced-GJK** [4], which improves the robustness of the original algorithm. In parallel, the Minkowski Portal Refinement (**MPR**) algorithm [5] [6] uses the Minkowski transformation to find intersections in the configuration space, but this algorithm also has some of the limitations of GJK.

The Expanding Polytope Algorithm (**EPA**) [7] used in combination with GJK, manages to overcome these difficulties, although at the cost of increased computational complexity.

Despite the effectiveness of these methods, one of the main limitations of traditional systems is their *non-differentiability*, which makes them unsuitable for gradient-based optimization approaches.

To address these issues, differentiable approaches, such as **smoothed GJK** [8], have been introduced. This method uses randomization techniques to compute approximate gradients. However, it is still computationally expensive and not always robust to object penetration.

In this scenario, the **DCOL** algorithm [1] addresses these limitations by reformulating collision detection as a convex optimization problem that computes the minimum uniform scaling required for the intersection of two primitives. The main advantages of this approach are its generality, its ability to handle a wide range of geometric primitives, and its differentiability, which allows the computation of precise gradients. These strengths make DCOL a versatile and differentiable solution for collision detection on a wide range of primitives.

2.2 Convex optimization problem

The concept of convex optimization plays a central role in this work, as it represents the theoretical basis of the DCOL Algorithm, described in chapter 3. What distinguishes convex optimization from other optimization problems is the fact that, under appropriate conditions, the search for the global minimum is reflected in the search for the local minimum.
Let us first consider a general optimization problem.
An optimization problem can be described as the process of finding the minimum (or maximum) of an objective function $f(x)$ subject to a set of constraints. Formally, an optimization problem takes the following form:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &subject \quad to \quad x \in \mathcal{C} \end{aligned} \tag{1}$$

where $f(x) : \mathbb{R}^n \to \mathbb{R}$ is the objective function to be optimized and $\mathcal{C} \subseteq \mathbb{R}^n$ represents the set of constraints, i.e., the admissible space of solutions. The optimal solution x^* is the solution that is associated with the minimum value of the objective function respecting all the constraints of the problem.
An optimization problem is said to be **convex** if:

- the objective function $f(x)$ is convex[1];
- the set of constraints $\mathcal{C}$ is a convex set [2].

The peculiarities of convex problems lie in a series of properties that differentiate them from other optimization problems. These properties are:

1. *Uniqueness of Solution*: If $f(x)$ is a strictly convex function, the optimal solution is unique.
2. *Computational Efficiency*: Convex problems can be solved with efficient algorithms, such as gradient methods, interior point method, or conic programming solvers, introduced in the next section.
3. *Robustness*: The convex structure guarantees that any local minimum is also a global minimum.

[1] From a geometric point of view, a function is said to be convex if, for each pair of points in its domain, the chord that connects them is above the graph of the function itself.

[2] A set $\mathcal{C} \subseteq \mathbb{R}^n$ is said to be convex if the segment that connects any two points belonging to $\mathcal{C}$ lies entirely in $\mathcal{C}$.

2.3 Conic Optimization and Differentiating Through a Cone Program

A conic optimization problem is a subclass of convex optimization problems that can be represented in the following form [9]:

$$\begin{aligned} \min_{x} \quad & c^\top x \\ \text{subject to} \quad & h - Gx \in \mathcal{K}, \end{aligned} \tag{2}$$

with $x \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, $G \in \mathbb{R}^{m \times n}$ and $h \in \mathbb{R}^m$. $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_N$ is the Cartesian product of N convex cones.
The characteristic of conic optimization lies in the constraints, which require that $h - Gx$ belongs to the convex cone $\mathcal{K}$, rather than to a generic convex set. The optimality conditions for the above conic optimization problem are:

$$\begin{aligned} c + G^T z &= 0 \\ h - Gx &\in \mathcal{K} \\ z &\in \mathcal{K}^* \\ (h - Gx) \circ z &= 0 \end{aligned} \tag{3}$$

where $z \in \mathbb{R}^m$ is a dual variable [3] associated with the conical constraints and $\mathcal{K}^*$ is the dual cone of $\mathcal{K}$ [10].
With the evolution of optimization techniques, conic problems have been extended to allow for *differentiation with respect to problem parameters*, such as c, G, and h. The basis of this approach is the **implicit function theorem**, which describes a relationship between the solutions and the problem parameters. The basis of this process is the concept of an implicit function, defined as:

$$g(y^*, \theta) = 0 \tag{4}$$

where:

- y^* is an equilibrium point;
- $\theta \in \mathbb{R}^b$ represents the parameters of the problem.

The implicit function theorem allows us to calculate the variations of the solution y with respect to the parameters θ through the implicit derivative. Expanding g in Taylor series to the first order and solving:

$$\frac{\partial g}{\partial y}\delta y + \frac{\partial g}{\partial \theta}\delta\theta = 0 \tag{5}$$

[3] The dual variable z quantifies the influence of the constraints on the optimal value of the objective function. In particular, z belongs to the dual cone associated with the constraints and satisfies a complementarity relation, indicating whether a constraint is active in the optimal solution.

we obtain:

$$\frac{\partial y}{\partial \theta} = -\left(\frac{\partial g}{\partial y}\right)^{-1} \frac{\partial g}{\partial \theta} \tag{6}$$

In this context, the optimality conditions of the conic problem (primal and dual constraints) are treated as an implicit function, allowing for the efficient computation of the derivatives of the optimal solutions.
When the conic problem is solved with a *primal-dual interior-point* method [11] [12], the computations can be performed directly during the solution phase, without requiring additional matrix factorizations [13]. This ensures high computational efficiency.
In the case where only the gradient of the objective function J with respect to the parameters θ is required, the implicit function theorem is not needed. In this case, it is sufficient to compute the gradient of the Lagrangian:

$$\mathcal{L}(x, z, \theta) = c(\theta)^T x + z^T (G(\theta)x - h(\theta)) \tag{7}$$

where c, h and G depend on the parameters θ. The gradient of the objective function is then:

$$\nabla_\theta J = \nabla_\theta \mathcal{L}(x^*, z^*, \theta) \tag{8}$$

where x^* and z^* are the primal and dual solutions respectively. This approach reduces the computational complexity compared to using the implicit function.

2.3.1 Second-order cone programming (SOCP)

A second-order cone (SOC), often called the Lorentz cone or ice-cream cone, is a set of vectors in $\mathbb{R}^n$ that satisfy a constraint relating one (scalar) component to the Euclidean norm of the remaining components. In $\mathbb{R}^n$, an SOC takes the form [14]

$$\mathcal{K} \;=\; \left\{ (x_0, x_1) \;\in\; \mathbb{R} \;\times\; \mathbb{R}^{n-1} \;\middle|\; x_0 \;\geq\; \|x_1\|_2 \right\} \tag{9}$$

Where:

- $x_0 \in \mathbb{R}$ is a scalar (the "top" component),
- $x_1 \in \mathbb{R}^{n-1}$ is a vector (the "tail"),
- $\|x_1\|$ denotes the Euclidean (L2) norm of x_1.

Essentially, a SOC requires that the scalar part x_0 be at least as large as the length (norm) of the vector part x_1. This geometric shape resembles an open, ice-cream-cone-like region in $\mathbb{R}^n$ with its apex at the origin. SOCs are central to Second-Order Cone Programming (SOCP). In these problems, one minimizes (or maximizes) a linear/quadratic objective subject to constraints in the form of linear equations/inequalities plus one or more second order cones. The combination of linear constraints and second-order constraints fits neatly into certain modern convex optimization frameworks, which is why SOCs appear in many geometry, control, and signal-processing applications.

3 The DCOL algorithm

After introducing the idea behind DCOL and its advantages over traditional techniques for calculating collisions, in this chapter we intend to describe in detail how it works.

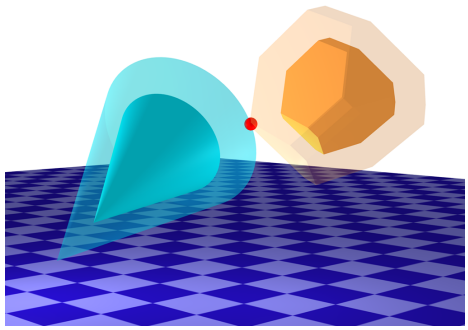

Figure 1: Collision detection between a cone and a polytope.

The DCOL algorithm is based on the resolution of a convex optimization problem, which allows to determine the minimum uniform scaling factor $\alpha \in \mathbb{R}$, applied simultaneously to two convex objects, necessary for a collision to occur between them.
In particular, the value of α is greater than 1 when the objects are not in collision and less than 1 when interpenetration occurs.
The optimization problem is defined as follows:

$$\begin{aligned} \min_{x,\alpha} \quad & \alpha \\ & x \in \mathcal{S}_1(\alpha), \\ s.t. \quad & x \in \mathcal{S}_2(\alpha), \\ & \alpha \geq 0. \end{aligned} \tag{10}$$

The set $\mathcal{S}(\alpha)$ represents a convex primitive scaled by a factor of α, where a point $x \in \mathbb{R}^3$ belongs to $\mathcal{S}(\alpha)$ if it is inside the scaled primitive.
Therefore, the constraint $x \in \mathcal{S}_1(\alpha)$ guarantees that the point x belongs to the first object scaled by a factor of α, as well as the second, $x \in \mathcal{S}_2(\alpha)$, guarantees the same on the second primitive.
The effectiveness of DCOL comes from the properties of this optimization problem. Its main properties are:

- *Convexity*: this guarantees the uniqueness of the solution.
- *Boundedness*: thanks to the constraint on α ($\alpha \geq 0$).

- *Feasibility*: this comes from the fact that each object is uniformly scaled, therefore, for sufficiently large values of α, the intersection becomes inevitable.

An important aspect that needs to be emphasized is the choice to scale the objects uniformly.
This strategy arises from the idea of transforming the collision problem into a continuous optimization problem. Instead of simply checking whether two objects intersect in their current configuration, we look for the minimum value of α that leads to an intersection between their scaled versions. When two objects do not intersect, their state can be described in terms of "how far away they are from starting to collide". Scaling the objects allows us to reduce this distance virtually, without altering the geometry or the relative positioning of the objects. One of the main advantages of using α is that this parameter does not depend on the specific shape of the objects or their geometric configuration. Furthermore, scaling the objects uniformly allows us to keep the convexity of the primitives intact, allowing us to formulate the problem as a convex optimization problem.
In the following, we report in detail the primitives used by the DCOL algorithm.

3.1 Primitives

The key idea of DCOL lies in the definition of geometric primitives, which are fundamental tools for modeling three-dimensional objects in space. These primitives are used both to describe the surrounding environment and to represent the robot whose trajectory we want to optimize.
Geometric primitives are basic shapes — polytopes, capsules, cylinders, cones, ellipsoids, spheres, and padded polygons — that, thanks to their integrability and flexibility, are able to represent a wide range of complex objects, making them extremely suitable for representing real-world environments.
The primitives were selected for their convex nature, which plays a crucial role in formulating collision problems as convex optimization problems. This property ensures that the solution to the problem is computationally tractable and can be solved robustly using efficient optimizers.
Each primitive is described mathematically by a set of well-defined constraints, which guarantee two essential properties:

- Convexity of constraints: This allows to solve the collision problem using convex optimizers, ensuring stability and computational efficiency.

- Derivability: Fundamental for computing gradients.

In the context of DCOL, primitives are used to calculate collisions between objects and to determine the minimum distance or degree of interpenetration between them. Each collision between two primitives is modeled as an optimization problem that searches for the minimum scaling factor α, necessary for the primitives to collide.

Scaling factor	Meaning
$\alpha > 1$	Primitives do not collide.
$\alpha = 1$	The primitives are in contact.
$\alpha < 1$	The primitives intersect.

Table 1: Minimum scaling factor, α.

The following sections present the mathematical constraints associated with each of the six primitives supported by DCOL.

3.1.1 Polytope

The polytope is a convex shape in three-dimensional space ($\mathbb{R}^3$) defined as the set of points $\omega \in \mathbb{R}^3$ that satisfy a system of half-space constraints [4].
The constraints are represented as:

$$A\omega \leq b \tag{11}$$

where each row of $A \in \mathbb{R}^{m\times 3}$ (Matrix describing m half-planes) and the corresponding value in $b \in \mathbb{R}^m$ (vector determining the relative positions of the half-planes in space) define a plane, and the polytope is the convex intersection of all the half-spaces defined by these planes; while $\omega \in \mathbb{R}^3$ is a point in space expressed in a local reference system $\mathbb{B}$.
The polytope can be uniformly scaled by applying a scaling parameter $\alpha > 0$. This scaling parameter allows to "move away" or "move closer" the half-planes with respect to the center of the polytope, maintaining the same convex shape. The constraint to check if a point x is inside the scaled polytope is:

$$AQ^T(x - r) \leq \alpha b \tag{12}$$

where Q is the rotation matrix connecting the polytope's reference frame $\mathbb{B}$ with the global frame $\mathbb{W}$, r is the origin of the local frame $\mathbb{B}$, and x is the coordinate of the point in global space.
We can rewrite the constraint in standard form [5]:

$$h_{ort} - G_{ort}\begin{bmatrix} x \\ \alpha \end{bmatrix} = AQ^T r - \begin{bmatrix} AQ^T & -b \end{bmatrix}\begin{bmatrix} x \\ \alpha \end{bmatrix} \in \mathbb{R}_+ \tag{13}$$

The G_{ort} and h_{ort} notation will be more clear in Section 3.2.

[4] The half-space constraints describe the edges of the polytope as the intersection of planes.
[5] The standard form allows treating the constraint as part of a convex optimization problem

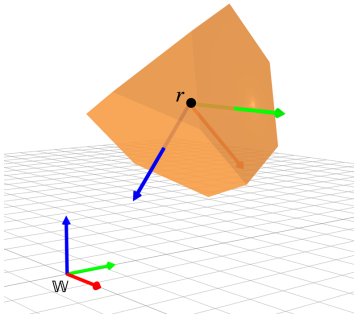

Figure 2: Polytope.

3.1.2 Capsule

A capsule is a three-dimensional convex shape defined as the set of points that lie within a certain distance R of a line segment. This segment lies along the x axis of a local reference frame $\mathbb{B}$. It has a length L, with its endpoints at a distance $L/2$ from the center of the segment. In simple terms, the capsule is a sort of "cylinder" with rounded ends.
To check whether a point x lies inside the capsule scaled by the parameter α, two constraints must be satisfied (14 & 15):

$$||x - (r + \gamma \hat{b}_x)||_2 \leq \alpha R \tag{14}$$

where r is the center of the capsule segment, γ is the slack variable indicating the position along the segment (for example $\gamma = 0$ is the center and $\gamma = \pm L/2$ are the endpoints) and $\hat{b}_x = Q[1, 0, 0]^T$.

$$-\alpha \frac{L}{2} \leq \gamma \leq \alpha \frac{L}{2} \tag{15}$$

This constraint requires that the variable γ does not exceed the bounds of the scaled segment.
The constraints in standard form are:

$$h_{ort} - G_{ort} \begin{bmatrix} x \\ \alpha \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0_{1\times 3} & -L/2 & 1 \\ 0_{1\times 3} & -L/2 & -1 \end{bmatrix} \begin{bmatrix} x \\ \alpha \\ \gamma \end{bmatrix} \in \mathbb{R}^2_+ \tag{16}$$

$$h_{soc} - G_{soc} \begin{bmatrix} x \\ \alpha \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ -r \end{bmatrix} - \begin{bmatrix} 0_{1\times 3} & -R & 0 \\ -I_3 & 0_{3\times 1} & \hat{b}_x \end{bmatrix} \begin{bmatrix} x \\ \alpha \\ \gamma \end{bmatrix} \in \mathbb{Q}_4 \tag{17}$$

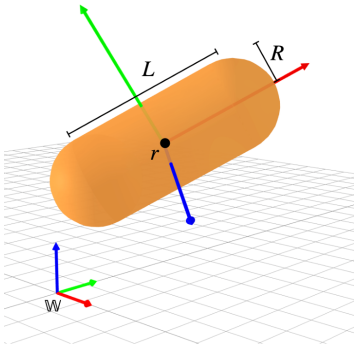

Figure 3: Capsule.

3.1.3 Cylinder

The constraints used to represent a cylinder of radius R and length L are the same as those used to represent the capsule (14 and 15), with the addition of the following constraints (which define the two flat faces of the cylinder at its ends):

$$\begin{aligned} \left[x - (r - \alpha\tfrac{L}{2}\hat{b}_x)\right]^T \hat{b}_x \geq 0 \\ \left[x - (r + \alpha\tfrac{L}{2}\hat{b}_x)\right]^T \hat{b}_x \leq 0 \end{aligned} \tag{18}$$

where the quantities $r \pm \alpha\frac{L}{2}b_x$ represent the centers of the two flat faces of the cylinder.
The inequalities guarantee that the points x are inside the two faces of the cylinder. In standard form, the constraints take the following form:

$$\begin{bmatrix} -\hat{b}_x^T r & -\frac{L}{2} & 0 \\ \hat{b}_x^T r & -\frac{L}{2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \alpha \\ \gamma \end{bmatrix} \in \mathbb{R}^2_+ \tag{19}$$

which overall results in the following constraints:

$$h_{ort} - G_{ort} \begin{bmatrix} x \\ \alpha \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -\hat{b}_x r \\ \hat{b}_x r \end{bmatrix} - \begin{bmatrix} 0_{1\times 3} & -L/2 & 1 \\ 0_{1\times 3} & -L/2 & -1 \\ -\hat{b}_x & -L/2 & 0 \\ \hat{b}_x & -L/2 & 0 \end{bmatrix} \begin{bmatrix} x \\ \alpha \\ \gamma \end{bmatrix} \in \mathbb{R}^2_+ \tag{20}$$

$$h_{soc} - G_{soc} \begin{bmatrix} x \\ \alpha \\ \gamma \end{bmatrix} = \begin{bmatrix} 0 \\ -r \end{bmatrix} - \begin{bmatrix} 0_{1\times 3} & -R & 0 \\ -I_3 & 0_{3\times 1} & \hat{b}_x \end{bmatrix} \begin{bmatrix} x \\ \alpha \\ \gamma \end{bmatrix} \in \mathbb{Q}_4 \tag{21}$$

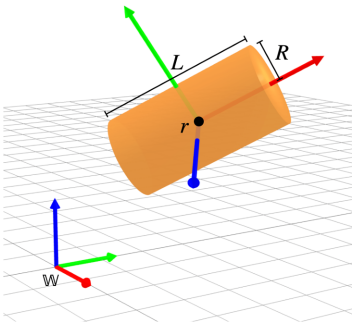

Figure 4: Cylinder.

3.1.4 Cone

The cone is described by:

- Height H : indicates the vertical distance from the base to the tip of the cone.
- Half angle β : is the angle formed between the axis of the cone and its lateral surface.
- Center point r : is the center point along the axis of the cone, located at a quarter of the height $\left(\frac{H}{4}\right)$) from the flat face towards the tip.

The constraints of the cone are given by two main equations:

1. Lateral constraint

$$||\tilde{x}_{2:3}||_2 \leq tan(\beta)\tilde{x}_1 \tag{22}$$

   Where $\tilde{x}$ is the point in space transformed through rotation and translation ($\tilde{x} = Q^T(x - r + \alpha\frac{3H}{4}\hat{b}_x)$), $\tilde{x}_{2:3}$ are the y, z components of the transformed point, and $\tilde{x}_1$ is the x component of the transformed point.
   This constraint ensures that the points lie inside the cone, by limiting their radial distance ($||\tilde{x}_{2:3}||$) as a function of their height along $\tilde{x}_1$. The term $tan(\beta)$ describes the opening angle of the cone.

2. Height constraint

$$\left(x - r - \alpha\frac{H}{4}\hat{b}_x\right)^T \hat{b}_x \leq 0 \tag{23}$$

   Where the term $\alpha\frac{H}{4}$ represents the scaling that governs the distance between the center of the flat base and the tip.
   This constraint requires that the points be below the plane of the flat base of the cone, limiting the cone vertically.

The constraints in standard form take the following form:

$$\begin{aligned} h_{ort} - G_{ort}\begin{bmatrix} x \\ \alpha \end{bmatrix} &= \hat{b}_x^T r - \begin{bmatrix} \hat{b}_x^T & -\frac{H}{4} \end{bmatrix}\begin{bmatrix} x \\ \alpha \end{bmatrix} \in \mathbb{R}_+^m \\ -h_{soc} - G_{soc}\begin{bmatrix} x \\ \alpha \end{bmatrix} &= -EQ^T r - \begin{bmatrix} -EQ^T & v \end{bmatrix}\begin{bmatrix} x \\ \alpha \end{bmatrix} \in \mathbb{Q}_3 \end{aligned} \tag{24}$$

Where $E = diag(tan\beta,\ 1,\ 1)$ and $v = (-\frac{3H}{4}tan\beta,\ 0,\ 0)$.

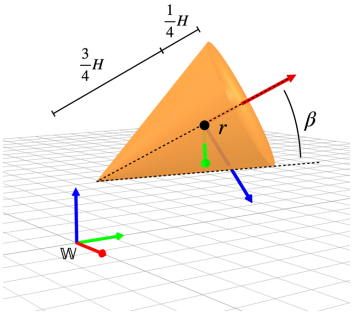

Figure 5: Cone.

3.1.5 Ellipsoid

An ellipsoid is a three-dimensional geometric shape defined by the following quadratic inequality:

$$x^T P x \leq 1 \tag{25}$$

where $P \in \mathbb{S}_{++}^n$ is a positive definite symmetric matrix describing the shape and orientation of the ellipsoid. The matrix P can be decomposed by the Cholesky factor $U \in \mathbb{R}^{n \times n}$ which allows to simplify the representation of the ellipsoid.
The constraint to be able to represent a scaled ellipsoid with arbitrary position and orientation is:

$$||UQ^T(x - r)||_2 \leq \alpha \tag{26}$$

where UQ^T is a matrix that combines the Cholesky factor U with the rotation Q^T, allowing to orient the ellipsoid in the global space and r is the center of the ellipsoid.
An ellipsoid can become a sphere simply by choosing $P = I/R^2$, with R being the radius of the sphere. In this case, the constraint reduces to checking the Euclidean distance from the center.
In standard form, the constraint takes the following form:

$$h_{soc} - G_{soc}\begin{bmatrix} x \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ -UQ^T r \end{bmatrix} - \begin{bmatrix} 0_{1\times 3} & -1 \\ -UQ^T & 0_{3\times 1} \end{bmatrix}\begin{bmatrix} x \\ \alpha \end{bmatrix} \in \mathbb{Q}_4 \tag{27}$$

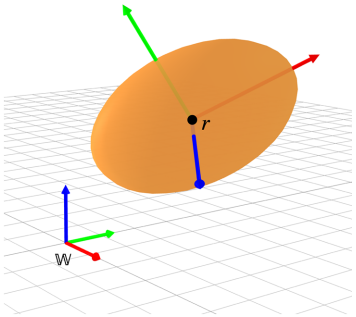

Figure 6: Ellipsoid.

3.1.6 Padded Polygon

A padded polygon is a padded two-dimensional polygon, that is, the set of points in three-dimensional space that are within a certain distance R from a two-dimensional polygon.
To check whether a point $x \in \mathbb{R}^3$ belongs to the padded polygon scaled by the parameter α, two constraints must be satisfied (28 & 29):

$$||x - (r + \tilde{Q}y)||_2 \leq \alpha R \tag{28}$$

where r is the origin of the polygon in the global reference frame, $y \in \mathbb{R}^2$ is the slack variable that represents a position inside the two-dimensional polygon, and $\tilde{Q} \in \mathbb{R}^{3\times 2}$ is the matrix that contains the first two columns of Q.

$$Cy \leq \alpha d \tag{29}$$

where $C \in \mathbb{R}^{m\times 2}$ is the half-space constraint matrix of the polygon and $d \in \mathbb{R}^m$ is the vector determining the relative position of the half-planes.
This constraint ensures that y belongs to the two-dimensional polygon, uniformly scaled by the parameter α.
In standard form, the constraints take the following form:

$$h_{ort} - G_{ort} \begin{bmatrix} x \\ \alpha \\ y \end{bmatrix} = 0_m - \begin{bmatrix} 0_{m\times 3} & -d & C \end{bmatrix} \begin{bmatrix} x \\ \alpha \\ y \end{bmatrix} \in \mathbb{R}^m_+ \tag{30}$$

$$h_{soc} - G_{soc} \begin{bmatrix} x \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ -r \end{bmatrix} - \begin{bmatrix} 0_{1\times 3} & -R & 0_{1\times 2} \\ -I_3 & 0_{3\times 1} & \tilde{Q} \end{bmatrix} \begin{bmatrix} x \\ \alpha \end{bmatrix} \in \mathbb{Q}_4 \tag{31}$$

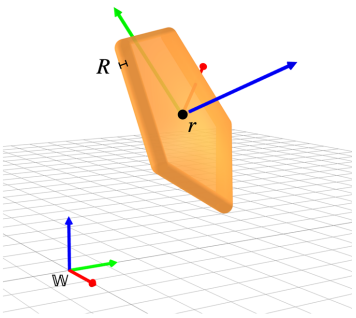

Figure 7: Padded Polygon.

3.2 Primal-Dual Interior Point Method

To solve 10, we use a PDIP method. As described in [11] and [12], a PDIP solve conic program of the form:

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & Gx + s = h, \\ & s \in \mathcal{K} \end{aligned} \tag{32}$$

That differs from Equation 2 for the usage of the slack variables s, living in some cone $\mathcal{K}$. In this context, $\mathcal{K}$ is a combination nonnegative orthants and second-order cones. The solution $x = \begin{pmatrix} \alpha & c \end{pmatrix}^\top \in \mathbb{R}^4$ contains both the proximity distance α and the coordinates of the contact point. G and h are computed in the following way:

- For both primitives involved in the collision detection, the problem matrices G_{ort}, G_h, G_{soc}, h_{soc} are computed using their geometrical properties. Details on these problem matrices are given in 3.1.
- The problem matrices are then combined in order to obtain a single constraint matrix G and a bound vector h

3.2.1 Overview

PDIP start from a feasible initial guess of the solution $\begin{pmatrix} x^{(0)} & s^{(0)} & z^{(0)} \end{pmatrix}$ and iteratively improve it by solving, at each iteration, the linearization of the Karush-Kuhn-Tucker conditions, in order to find the step directions Δx, Δs, Δz. The KKT conditions are the following:

1. Primal Feasibility:
$$Gx + s = h$$
2. Dual Feasibility:
$$G^T z + c = 0$$

3. Complementarity condition:

$$s \circ z = 0$$

With $\circ$ denoting the elementwise product, enforcing that s_i or z_i must be 0 for each constraint. Actually, we use the relaxed centrality condition:

$$s \circ z = \mu e$$

since $\mu \to 0$ over iterations, it increasingly satisfy complementarity.

Line-search and Nesterov-Todd Scaling are implemented and described in Appendix B and C. Alle the formulae of the algorithm are described below for completeness. Note that $\circ$ indicates the cone product, described in Appendix A, while *Chol* indicates the Cholesky decomposition, described in D. The pseudo-algorithm of PDIP is described in Algorithm 1.

Algorithm 1 Primal-Dual Interior Point Method (PDIP)

procedure PDIP($c, G, h, \mathrm{M}, \mathrm{tol}$)
  Initialize (x, s, z) within the feasible cone
  for $k = 1, \ldots, \mathrm{M}$ **do**
    $W \leftarrow \mathrm{NTScaling}(s, z)$
    Compute residuals r_x, r_s, μ
    if $\mu < \mathrm{tol}$ **then**
      return (x, s, z)
    end if
    Affine Step:
      Solve KKT system for search directions $(\Delta x, \Delta s, \Delta z)$,
      $\alpha \leftarrow$ Perform line search on $(\Delta s, \Delta z)$ to ensure feasibility
    Correction Step:
      Compute the corrections with σ
      Compute corrected $(\Delta x, \Delta s, \Delta z)$ and line-search α
    Update variables:
      $x \leftarrow x + \alpha \Delta x$
      $s \leftarrow s + \alpha \Delta s$
      $z \leftarrow z + \alpha \Delta z$
  end for
  if here: the algorithm failed to converge
end procedure

Newton Direction Computation

1. Duality Gap:

$$\mu = \frac{\mathbf{s}^\top \mathbf{z}}{n},$$

where n is the total cone degree.

2. Primal and Dual Residuals:

$$\mathbf{r}_x = \mathbf{G}^\top \mathbf{z} + \mathbf{c}, \quad \mathbf{r}_z = \mathbf{s} + \mathbf{G}\mathbf{x} - \mathbf{h}.$$

measuring how much the current solution violates the KKT conditions.

3. Scaled Variables:

$$\lambda = W\mathbf{z}, \quad \lambda\lambda = \lambda \circ \lambda.$$

4. Affine Newton Directions:

$$\begin{aligned} \mathbf{b}_x &= -\mathbf{r}_x, & \lambda_{\Delta s} &= \lambda^{-1} \circ (-\lambda\lambda), \\ \mathbf{b}_{\tilde{z}} &= W^{-1}\left(-\mathbf{r}_z - W\lambda_{\Delta s}\right), & \tilde{\mathbf{G}} &= W^{-1}\mathbf{G}. \end{aligned}$$

5. Solve for $\Delta\mathbf{x}$:

$$\mathbf{F} = \mathit{Chol}(\tilde{\mathbf{G}}^\top \tilde{\mathbf{G}}), \quad \Delta\mathbf{x} = \mathbf{F}^{-1}\left(\mathbf{b}_x + \tilde{\mathbf{G}}^\top \mathbf{b}_{\tilde{z}}\right).$$

6. Solve for $\Delta\mathbf{z}$ and $\Delta\mathbf{s}$:

$$\Delta\mathbf{z} = W^{-1}\left(\tilde{\mathbf{G}}\Delta\mathbf{x} - \mathbf{b}_{\tilde{z}}\right), \quad \Delta\mathbf{s} = W\left(\lambda_{\Delta s} - W\Delta\mathbf{z}\right).$$

Corrector Term:

$$\sigma = \left(\frac{(\mathbf{s} + \alpha\Delta\mathbf{s})^\top (\mathbf{z} + \alpha\Delta\mathbf{z})}{\mathbf{s}^\top \mathbf{z}}\right)^3$$

Measuring complementarity of updated variables compared to current ones. Used then to update the directions:

$$\begin{aligned} \mathbf{d}_s &= -\lambda\lambda - \left(W^{-1}\Delta\mathbf{s} \circ W\Delta\mathbf{z}\right) + \sigma\mu\mathbf{e}, \\ \lambda_{\Delta s} &= \lambda^{-1} \circ \mathbf{d}_s \\ b_{\tilde{z}} &= W^{-1}(-r_z - W\lambda_{\Delta s}) \end{aligned} \tag{33}$$

Then Δx, Δs and Δz are re-computed using the same formula as above, but now the values are corrected.

4 ALTRO algorithm

In the following section, we present an overview of the *Augmented Lagrangian TRajectory Optimizer* (ALTRO) algorithm, a powerful method for solving trajectory constrained optimization problems [2].
Existing techniques in the literature can be divided into two main categories:

- *Direct methods*: they treat both state x and control input u as decision variables and uses general-purpose nonlinear programming solvers (NLP), such as IPOPT.
- *Indirect methods*: they exploit the Markovian structure [6] of the problem and only treat the control inputs u as decision variables, with the dynamics constraints implicitly enforced by simulating the dynamics of the system. For instance, iLQR, is an indirect method that breaks the problem into a sequence of smaller sub-problem.

The strength of ALTRO lies in its ability to combine the advantages of both direct and indirect methods, ensuring speed, numerical robustness, and the ability to handle complex constraints. The general optimization problem to be solved is:

$$\begin{aligned} \min_{x_0:N, u_0:N-1} \; & l_f(x_N) + \sum_{k=0}^{N-1} l(x_k, u_k) \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k), \\ & g_k(x_k, u_k) \leq 0, \\ & h_k(x_k, u_k) = 0, \end{aligned} \tag{34}$$

where l_f and l arew the final and stage cost functions, x_k and u_k are the state and input control variables, $f(x_k, u_k)$ is the discrete dynamics function, and $g(x_k, u_k)$ and $h(x_k, u_k)$ are inequality and equality constrains, respectively.
Let's now understand the refinements and extensions of ALTRO, and then provide a pseudo-code.

Infeasible State Trajectory Initialization: A common challenge in trajectory optimization is the difficulty of determining an initial control sequence that generates a dynamically feasible state trajectory. In other words, even when one knows where the system should go, it is not always obvious how to drive it there while respecting physical laws and system constraints. To address this difficulty, ALTRO introduces artificial controls $w_k \in \mathbb{R}^n$ which allow the system to gradually adapt toward an optimal and feasible solution. Specifically, the discrete system dynamics are modified to include this auxiliary control:

$$x_{k+1} = f(x_k, u_k) + \omega_k \tag{35}$$

[6] x_{k+1} depends only on (x_k, u_k)

Given the initial state $\tilde{X}$ and the control trajectories U, the infeasible controls $W = \{\omega_0, \cdots, \omega_{N-1}\}$ are computed as the difference between the real and desired dynamics

$$\omega_k = \tilde{x}_{k+1} - f(x_k, u_k) \tag{36}$$

The addition of ω_k modifies the original optimization problem (34), by introducing an extra cost term

$$\sum_{k=0}^{N-1} \frac{1}{2} \omega_k^T R_{inf} \omega_k \tag{37}$$

and constraints $\omega_k = 0$, $k = 0, \cdots, N-1$. These auxiliary controls are gradually eliminated during the optimization process, ensuring that the solution to the modified problem converges to the solution of the original problem.

Minimum Time: ALTRO also handles total time minimization problems. This goal is achieved by introducing $\tau_k = \sqrt{dt} \in \mathbb{R}$ as an input at each time step—an additional control that represents the discrete time interval at each step—thus optimizing the time trajectory $T = \{\tau_0, \cdots, \tau_{N-1}\}$. The original problem (34) is modified as follows

$$\begin{bmatrix} x_{k+1} \\ \omega_{k+1} \end{bmatrix} = \begin{bmatrix} f(x_k, u_k, \tau_k) \\ \tau_k \end{bmatrix} \tag{38}$$

with the added cost

$$\sum_{k=0}^{N-1} R_{min} \tau_k^2 \tag{39}$$

and constraints $\omega_k = \tau_k$, $k = 1, \cdots, N-1$ which ensure that the optimized trajectories are consistent and accurately represent the real system's dynamic behavior

The Two Phases of ALTRO: The ALTRO algorithm consists of two main phases:

- First Phase: The algorithm uses the iterative Linear Quadratic Regulator (iLQR) method to quickly solve the unconstrained subproblems within the Augmented Lagrangian (AL) framework. This phase produces an approximate solution that meets the constraints with a relatively coarse tolerance
- Second Phase: The approximate solution obtained in the first phase is used as a starting point for a projected Newton method, designed to refine the solution and achieve high accuracy in satisfying the constraints. This phase is called "solution polishing.

The pseudo-code for ALTRO is shown in Algorithm 2. The solution obtained with AL-iLQR was good enough for our purposes, so we did not implement the second phase. In the following section, we will describe iLQR in details.

Algorithm 2 ALTRO

procedure ALTRO
  Initialize x_0, U, tolerances; $\tilde{X}$
  if Infeasible Start **then**
    $X \leftarrow \tilde{X}$, $W \leftarrow$ from (36)
  else
    $X \leftarrow$ Simulate from x_0 using U, (41), (35), (38)
  end if
  $(X, U), \lambda \leftarrow$ AL-iLQR$((X, U), \text{iLQR}, \text{tol.})$
  $(X, U) \leftarrow$ ProjectedNewton$((X, U, \lambda), \text{tol.})$
  return X, U
end procedure

4.1 iLQR

Before describing AL-iLQR, used in the first phase of ALTRO, we introduce the iterative Linear Quadratic Regulator (iLQR). This algorithm minimizes the following general cost function:

$$J(X, U) = l_f(x_N) + \sum_{k=0}^{N-1} l(x_k, u_k) \tag{40}$$

subject to the nonlinear dynamics

$$x_{k+1} = f(x_k, u_k) \tag{41}$$

where $x \in \mathbb{R}^n$ is the system state and $u \in \mathbb{R}^m$ is the control input. Constraints are handled using an initial state x_0 and a nominal control trajectory $U = \{u_0, \cdots, u_{N-1}\}$ to simulate the state trajectory $X = \{x_0, \cdots, x_N\}$. A pseudo-code for iLQR is provided in Algorithm 3.

Algorithm 3 Iterative LQR

Initialize x_0, U, tolerance
$X \leftarrow$ Simulate from x_0 using U, (41)
function iLQR(X, U)
  $J \leftarrow$ Using X, U, (40)
  repeat
    $J^- \leftarrow J$
    $K, d, \Delta V \leftarrow$ **BackwardPass**(X, U)
    $X, U, J \leftarrow$ **ForwardPass**$(X, U, K, d, \Delta V, J^-)$
  until $|J - J^-| \leq$ tolerance
  return X, U, J
end function

iLQR consists of two main steps:

- **Backward Pass**: The main objective of Backward Pass is to compute the optimal updates for controls and state trajectories to reduce the overall cost of the system. The underlying idea is to use a recursive backward approach, starting from the final time step (N) and progressing back to the initial step (0).
  Initially, the cost-to-go is computed as follows:

$$\begin{aligned} &V_N(x_N) = l_f(x_N) \\ &V_k(x_k) = \min_{u_k}\{l(x_k, u_k) + V_{k+1}(f(x_k, u_k))\} = \min_{u_k} Q_k(x_k, u_k) \end{aligned} \tag{42}$$

  where $Q_k(x_k, u_k)$ is the action-value function. To simplify the computation, the second-order Taylor expansion of $V_k(x_k)$ is applied:

$$\delta V_k(x_k) \approx p_k^T \delta x_k + \frac{1}{2}\delta x_k^T P_k \delta x_k \tag{43}$$

  where:

  - p_k is the gradient of $V_k(x_k)$: $p_k = \left.\frac{\partial V_k(x)}{\partial x}\right|_{x_k}$;
  - P_k is the Hessian matrix of $V_k(x_k)$: $P_k = \left.\frac{\partial^2 V_k(x)}{\partial x^2}\right|_{x_k}$.

  The values of p_k and P_k evaluated in the final state x_N initialize the recurrence to compute the optimal-to-go cost $V_k(x_k)$:

$$p_N = \left.\frac{\partial l_f(x)}{\partial x}\right|_{x_N} \tag{44}$$

$$P_N = \left.\frac{\partial^2 l_f(x)}{\partial x^2}\right|_{x_N} \tag{45}$$

  For each time step k, the derivatives of the action-value function with respect to the states and control inputs are computed. The second-order Taylor series expansion of Q_k is:

$$\delta Q_k = \frac{1}{2}\begin{bmatrix}\delta x_k \\ \delta u_k\end{bmatrix}^\top \begin{bmatrix}Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu}\end{bmatrix}\begin{bmatrix}\delta x_k \\ \delta u_k\end{bmatrix} + \begin{bmatrix}Q_x \\ Q_u\end{bmatrix}^\top \begin{bmatrix}\delta x_k \\ \delta u_k\end{bmatrix} \tag{46}$$

  where:

$$\begin{aligned} Q_{xx} &= \ell_{xx} + A_k^\top P_{k+1} A_k \\ Q_{uu} &= \ell_{uu} + B_k^\top P_{k+1} B_k \\ Q_{ux} &= \ell_{ux} + B_k^\top P_{k+1} A_k \\ Q_x &= \ell_x + A_k^\top p_{k+1} \\ Q_u &= \ell_u + B_k^\top p_{k+1} \end{aligned} \tag{47}$$

while A_k and B_k are the derivatives of the dynamics function with respect to the states and inputs.
At this point, Q_k is minimized with respect to δu_k to obtain a control law that corrects the nominal trajectory. This correction is expressed as a linear combination of the form:

$$\delta u_k^* = -(Q_{uu} + \rho I)^{-1}(Q_{ux}\delta x_k + Q_u) = K_k \delta x_k + d_k \tag{48}$$

where K_k is a feedback gain that adjusts the control based on the errors between the current and nominal states, d_k is a feedforward term that introduces direct corrections to the nominal control, independent of the current state, while ρ is a regularization term that guarantees the invertibility of Q_{uu}.
The last step consists in updating the vectors p_k and the matrices P_k using the following expressions:

$$\begin{aligned} p_k &= Q_x + K_k^\top Q_{uu} d_k + K_k^\top Q_u + Q_{xu} d_k \\ P_k &= Q_{xx} + K_k^\top Q_{uu} K_k + K_k^\top Q_{ux} + Q_{xu} K_k \end{aligned} \tag{49}$$

These updates allow to transfer the optimal cost information from the future steps $(k+1)$ to the previous ones (k). During each iteration, ΔV_k is also computed, representing the expected change in cost:

$$\Delta V_k = d_k^\top Q_u + \frac{1}{2} d_k^\top Q_{uu} d_k \tag{50}$$

This quantity is useful for evaluating the effectiveness of corrections and represents how much the total cost is expected to decrease at step k after applying the control correction.
After calculating:

1. the gradient p_k
2. the Hessian P_k
3. the expected change in cost ΔV_k

the whole process is repeated, step by step, until the first state x_0 is reached. This process provides the basic data for the Forward Pass. A pseudo-code for the Backward Pass is provided in Algorithm 4.

Algorithm 4 Backward Pass

1: **function** BACKWARDPASS(X, U)
2: $p_N, P_N \leftarrow$ (44), (45)
3: **for** $k = N-1 : -1 : 0$ **do**
4: $\delta Q \leftarrow$ (46), (47)
5: **if** $Q_{uu} > 0$ **then**
6: $K, d, \Delta V \leftarrow$ (48), (50)
7: **else**
8: Increase ρ and go to line 3
9: **end if**
10: **end for**
11: **return** $K, d, \Delta V$
12: **end function**

- **Forward Pass**: In the forward pass, the new state trajectory is computed starting from the control adjusted with the corrections calculated in the backward pass.

$$x = x_k, \quad u_k = u_k + \Delta u_k, \quad \Delta u_k = K_k \Delta x_k + d_k, \quad x_{k+1} = f(x_k, u_k)$$

where α is the step, computed with a simple line search. A pseudo-code of the forward pass is provided in Algorithm 5.

Algorithm 5 Forward Pass

1: **function** FORWARDPASS($X, U, K, d, \Delta V, J$)
2: Initialize $\bar{x}_0 = x_0$, $\alpha = 1$, $J^- \leftarrow J$
3: **for** $k = 0 : 1 : N-1$ **do**
4: $\bar{u}_k \leftarrow u_k + K_k(\bar{x}_k - x_k) + \alpha d_k$
5: $\bar{x}_{k+1} \leftarrow$ Using $\bar{x}_k, \bar{u}_k$, (3)
6: **end for**
7: $J \leftarrow$ Using X, U, (2)
8: **if** J satisfies line search conditions **then**
9: $X \leftarrow \bar{X}, U \leftarrow \bar{U}$
10: **else**
11: Reduce α and go to line 3
12: **end if**
13: **return** X, U, J
14: **end function**

4.2 Augmented Lagrangian iLQR (AL-iLQR)

The AuLA framework is described in Algorithm 6, in our case, since it is AL-iLQR, the SOLVER used by AuLA is precisely iLQR. The algorithm has two phases: in the first the Lagrangian is minimized using iLQR; in the second the penalty weights λ and μ are updated.

First Phase: The augmented Lagrangian of 34 is:

$$\begin{aligned}\mathcal{L}_A(X,U,\lambda,\mu) &= \ell_N(x_N) + \left(\lambda_N + \frac{1}{2} I_{\mu_N} c_N(x_N)\right)^\top c_N(x_N) \\ &+ \sum_{k=0}^{N-1} \ell_k(x_k,u_k,\Delta t) + \left(\lambda_k + \frac{1}{2} I_{\mu_k} c_k(x_k,u_k)\right)^\top c_k(x_k,u_k) \qquad (51) \\ &= \mathcal{L}_N(x_N,\lambda_N,\mu_N) + \sum_{k=0}^{N-1} L_k(x_k,u_k,\lambda_k,\mu_k)\end{aligned}$$

Where $\lambda_k \in \mathbb{R}^p$ is a Lagrange multiplier, $\mu_k \in \mathbb{R}^{p_k}$ is a penalty weight, and $c_k = (g_k, h_k) = \begin{pmatrix} g_k^\top & h_k^\top \end{pmatrix}^\top \in \mathbb{R}^p$ represents the concatenated set of equality and inequality constraints with active sets $\mathcal{I}_k$ and $\mathcal{E}_k$, respectively. I_{μ_k} is a diagonal matrix defined as:

$$I_{\mu_k,ii} = \begin{cases} 0, & \text{If} \quad c_{k_i}(x_k,u_k) < 0 \wedge \lambda_{k_i} = 0,\ i \in \mathcal{I} \\ \mu_{k,i}, & \text{otherwise} \end{cases}$$

This formulation ensures that:

- Inactive inequality constraints ($c_k \leq 0$) are not penalized
- Active constraints ($c_k \geq 0$) are penalized
- Equality constraints are always enforece

Where k_i denotes the i-th constraint at time step k. The dynamics constraints are handled implicitly using an initial state x_0 and a nominal control trajectory $U = \{u_0, \ldots, u_{N-1}\}$, to simulate forward the trajectory of state $X = \{x_0, \ldots, x_N\}$. The backward pass is derived by defining the optimal cost to be achieved for fixed penalty weights and multipliers, $V|_{\lambda,\mu}$, and the recursive relation:

$$V_N(x_N)|_{\lambda,\mu} = \mathcal{L}_N(x_N,\lambda_N,\mu_N)$$

$$\begin{aligned}V_k(x_k)|_{\lambda,\mu} &= \min_{u_k} \left(\mathcal{L}_k(x_k,u_k,\lambda_k,\mu_k) + V_{k+1}(f(x_k,u_k))|_{\lambda,\mu}\right) \\ &= \min_{u_k} Q(x_k,u_k)|_{\lambda,\mu}\end{aligned}$$

Where $Q_k = Q(x_k, u_k)|_{\lambda,\mu}$ is the action value function. To make the dynamic programming step traceable, we take a second-order Taylor expansion of V_k with respect to the fixed state variable and λ, μ:

$$\delta V_k \approx P_k^\top \delta x_k + \frac{1}{2}\delta x_k^\top P_k \delta x_k,$$

resulting in the optimal second-order expansion of the terminal cost-to-go

$$p_N = (\ell_x)_N + (\ell_{xx})_N \delta x_N + \frac{1}{2}(\ell_{xx})_N \delta x_N^2$$

$$P_N = (\ell_{xx})_N + (\ell_{xx})_N \delta x_N.$$

The relationship between δV_k and δV_{k+1} is established by taking the second-order Taylor expansion of Q_k with respect to state and control:

$$\delta Q_k = \frac{1}{2}\begin{bmatrix}\delta x_k \\ \delta u_k\end{bmatrix}^\top \begin{bmatrix}Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu}\end{bmatrix}\begin{bmatrix}\delta x_k \\ \delta u_k\end{bmatrix} + \begin{bmatrix}Q_x \\ Q_u\end{bmatrix}^\top \begin{bmatrix}\delta x_k \\ \delta u_k\end{bmatrix}$$

Omitting the time step indices for simplicity, the block matrices are:

$$Q_{xx} = \ell_{xx} + A^\top P^\top A + c_x^\top I_{\mu_k} c_x,$$

$$Q_{ux} = \ell_{ux} + B^\top P^\top A + c_u^\top I_{\mu_k} c_x,$$
$$Q_{uu} = \ell_{uu} + B^\top P^\top B + c_u^\top I_{\mu_k} c_u,$$
$$Q_x = \ell_x + A^\top p^\top + c_x^\top (\lambda_k + I_{\mu_k} c),$$
$$Q_u = \ell_u + B^\top p^\top + c_u^\top (\lambda_k + I_{\mu_k} c),$$

where $A = \partial f/\partial x$, $B = \partial f/\partial u$, and $\cdot'$ denotes the variables at step $k+1$. In line with the use of linearized dynamics in iLQR, the constraints are also linearized for the expansion.
Minimizing Q_k with respect to δu_k provides a correction to the control trajectory. The result consist of a feedforward term d_k and a linear feedback term $K_k \delta x_k$. Regularization is added to ensure the invertibility of Q_{uu}:

$$\delta u_k = -(Q_{uu} + \rho I)^{-1}(Q_{ux}\delta x_k + Q_u) = K_k \delta x_k + d_k.$$

Substituting δu_k in the back pass (7), we get a closed expression for p_k, P_k, and the expected cost ΔV_k:

$$P_k = Q_{xx} + K_k^\top Q_{uu} K_k + K_k^\top Q_{ux} + Q_{xu} K_k,$$
$$p_k = Q_x + K_k^\top Q_{uu} d_k + K_k^\top Q_u + Q_{xu} d_k,$$
$$\Delta V_k = d_k^\top Q_u + \frac{1}{2} d_k^\top Q_{uu} d_k.$$

A forward pass simulates the system using the correction to the nominal control trajectory and a linear search is performed on the feedforward term d_k to ensure cost reduction.

Second Phase: After an iLQR step with λ and μ fixed, the dual variables are updated according to:

$$\lambda_k^i = \begin{cases} \lambda_k^i + \mu_k^i c_k(x_k, u_k), & i \in \mathcal{E}_k \\ \max(0, \lambda_k^i + \mu_k^i c_k(x_k, u_k)), & i \in \mathcal{I}_k. \end{cases} \tag{52}$$

And the penalty weight is increased proportionally according to the rule:

$$\mu_k^i = \phi \mu_k^i,$$

where $\phi > 1$ is an update factor. The same procedure is repeated until the solution satisfies the required constraints, ensuring the convergence of the AL method towards a good solution.

Algorithm 6 Augmented Lagrangian

```
function AuLA(x_0, SOLVER, tolerance)
    Initialize λ, μ, φ
    while max(c) > tolerance do
        Minimize L_A(x, λ, μ) w.r.t. x using SOLVER
        Update λ using (23), update μ using (24)
    end while
    return X, λ
end function
```

5 Practical applications

In this chapter, we illustrate the usefulness and versatility of differentiable collision detection, demonstrating its effectiveness through three different application scenarios. In each of these, the main goal is to perform *trajectory optimization* while avoiding collisions. Ensuring that contact is strictly avoided is essential not only to ensure the operational safety of the systems, but also to optimize their overall efficiency.

Our approach is based on ALTRO and includes a dual optimization problem that integrates a collision constraint defined via the α parameter. This parameter ensures that the optimized trajectories strictly comply with the collision avoidance conditions. In particular, the dual optimization problem allows you to simultaneously manage the minimization of the main objectives and the respect of the safety constraints imposed by differentiable collision detection (DCOL). Each of the three implemented scenarios demonstrates how this approach can address complex problems and ensure optimal solutions that meet all imposed conditions. The ability to combine collision management with gradient-based optimization algorithms makes our model highly versatile and effective.
The applications presented in this chapter aim not only to showcase how robust and efficient our model is but also to highlight how useful it can be in real-world situations.

5.1 Trajectory Optimization

Trajectory optimization is a fundamental aspect in motion planning and control. It is a process that allows us to calculate the best path that a system can follow, taking into account its physical limitations and the conditions of the surrounding environment. The goal is to find a trajectory that not only satisfies all the imposed constraints, but at the same time optimizes a cost function.
A generic trajectory optimization problem with DCOL-based collision avoidance constraints can be formulated as follows:

$$\begin{aligned} \text{minimize} \quad & \ell_N(x_N) + \sum_{k=1}^{N-1} \ell_k(x_k, u_k) \\ \text{subject to} \quad & x_{k+1} = f_k(x_k, u_k), \\ & h_k(x_k, u_k) \leq 0, \\ & g_k(x_k, u_k) = 0, \\ & \alpha_k(x_k) \geq 1, \end{aligned}$$

Where $x = f(x, u)$ is the discrete dynamics of the systems[7], $h(x_k, u_k)$ indicates bounds on the state and control inputs (e.g. the torque must not overcome 200 Nm) and $g(x, u)$ imposes the initial and final states. Moreover, to ensure no interpenetration between the robot and obstacles, the collision constraint

[7] note that we have used RK4 integration

$\alpha(x) \geq 1$ is imposed at each timestep. This approach allows collision avoidance constraints to be integrated directly into the optimization process, ensuring that the calculated trajectories are safe and free of interpenetration between the geometric primitives considered.
We can solve this type of problems using ALTRO and DCOL, as we have explained in Sections 4 and 3. For all three applications above, the cost function is:

$$\ell_N(x_N) = \frac{1}{2}(x - x_g)^\top Q_f (x - x_g) \quad \text{for the terminal cost}$$

$$\sum_{k=1}^{N-1} \ell_k(x_k, u_k) = \frac{1}{2}(x - x_g)^\top Q (x - x_g) + \frac{1}{2}(u - u_g)^\top R (u - u_g) \quad \text{for the running cost}$$

Where Q_f, Q and R are weight matrices.

5.2 "Piano Mover" Problem

In the *Piano Mover* problem, a piano must navigate through a narrow corridor and make a 90-degree turn without colliding with the walls [15], [16]. This presents a significant challenge due to the size of the piano, *2.6 meters* long, and the small size of the corridor, which is only *1 meter* wide. To address this complexity, the piano is modeled as a two-dimensional cylindrical rigid body and the corridor is described through geometric obstacles represented by polytopes. The representation of the piano as a rigid body takes into account six state variables:

$$x = \begin{pmatrix} r & v & \theta & \omega \end{pmatrix}^\top \in \mathbb{R}^2 \times \mathbb{R}^2 \times \mathrm{SO}(2) \times \mathbb{R}$$

where:

- r represents the position (x, y) of the center of mass of the piano in the two-dimensional plane;
- v is a vector of 2 components comprising the linear velocities v_x and v_y
- θ is the angular orientation of the piano with respect to the axis x;
- ω represents the angular velocity

The control vector is instead made up of 3 variables:

$$u = \begin{pmatrix} a_v & a_\omega \end{pmatrix}^\top \in \mathbb{R}^2 \times \mathbb{R}$$

Where a_v represents the linear acceleration, while a_ω is the angular acceleration. The dynamics of the piano mover is a double integrator and it is the following:

$$\begin{aligned} \dot{r} &= v \\ \dot{v} &= a_v \\ \dot{\theta} &= \omega \\ \dot{\omega} &= a_\omega \end{aligned}$$

Each obstacle in the corridor is modeled as a rectangular prism, and the position and orientation of the piano are continuously compared to these obstacles to ensure there are no interpenetrations.
To determine an optimal trajectory, several constraints were considered. Among these, the parameter α plays a crucial role in ensuring collision avoidance. In addition to collision constraints, limits on the values of control states and commands have been applied to ensure that the system operates within physically realistic boundaries.
To avoid collisions with the walls of the corridor, the **DCOL** framework was used, while, to plan an optimal trajectory that allows the piano to reach the desired final configuration, the **ALTRO** framework was used. Thanks to the integration with DCOL, ALTRO calculates a trajectory that respects all the imposed constraints, while minimizing the cost function.
The analysis of the results obtained is supported by the images presented in Fig.8, Fig.9 and 10, which respectively illustrate the trajectory followed by the piano within the corridor, the evolution of control commands during the execution of the trajectory and the state trajectory of the Piano Mover. In the first image (Fig.8) we can see how the piano successfully avoids the corridor walls, confirming that the imposed collision constraints have been respected. The rotation of the piano is evident along the trajectory, especially when it navigates the 90-degree turn.

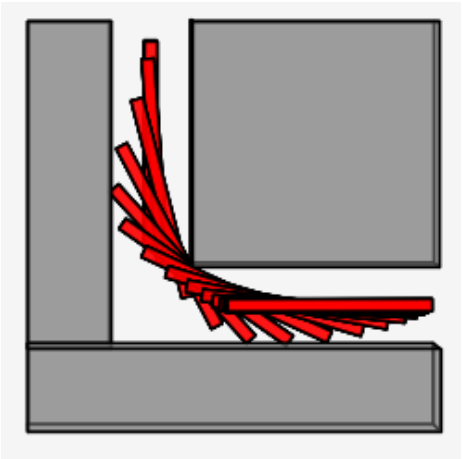

Figure 8: The piano maneuver

Fig.9 represents the evolution of the three control commands (linear accelerations along x and y and angular acceleration) during the execution of the trajectory. At the beginning, the control commands show significant variations to initiate the movement and address the critical phase of the curve, in particular, the angular acceleration undergoes an important initial variation and a negative

peak around the second $s = 2$. This is consistent with the moment when the piano must rotate quickly to fit the curve's geometry. After passing the critical phase of the curve (after the thirtieth time step), all control commands stabilize. This demonstrates that the system has managed to reach steady state as the piano approaches the desired final configuration.

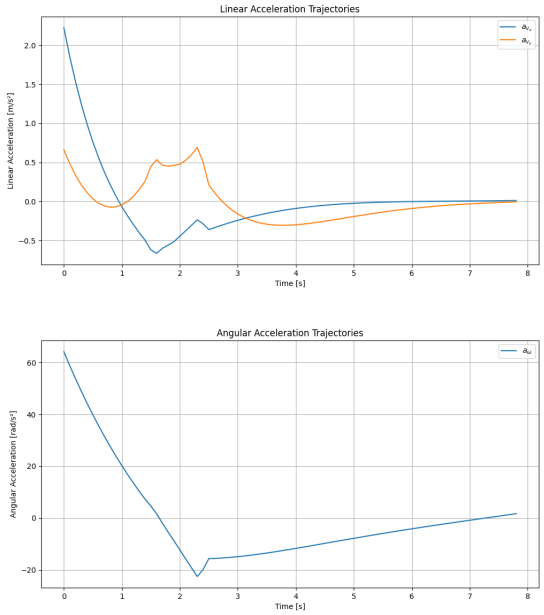

Figure 9: Control for the piano mover. Note how the angular accelerations are overcomed by the activity of the angular acceleration, that is the most relevant in this application

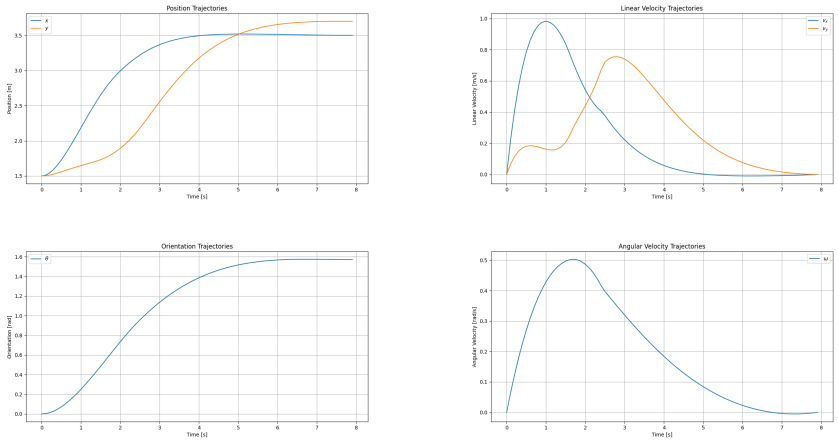

Figure 10: State trajectory of the Piano Mover experiment

The parameters used for this experiment are shown in Table 2.

Parameter	**Value**
timesteps (N)	80
dt	0.1
max number of iterations	3000
convergence tolerance (atol)	4×10^{-2}
max linesearch iterations	20
initial regulation	1×10^{-6}
min value for regulation	1×10^{-6}
max value for regulation	1×10^{2}
ρ	1
ϕ	10.0
constraint violation tolerance (convio)	1×10^{-4}
initial state (x_0)	[1.5, 1.5, 0, 0, 0, 0]
final state (x_g)	[3.5, 3.7, 0, 0, $\pi/2$, 0]
$u_{\min}$	-200 (element-wise)
$u_{\max}$	200 (element-wise)

Table 2: Parameters for the piano mover experiment.

The algorithm converged in 34 iterations of ALTRO. The time of execution for this experiment are available in Sect 6.
Thanks to the combination of ALTRO and DCOL, the Piano Mover problem was successfully addressed and solved, ensuring optimal trajectory planning and collision-free motion.

5.3 Quadrotor

The *Quadrotor* problem involves planning and controlling the trajectory of a quadricopter, an air vehicle with four rotors, in a complex three-dimensional environment populated by 12 obstacles.
The vehicle must move from an initial position x_0 to a final position x_g, following an optimal trajectory that ensures collision avoidance and respects dynamic and control constraints [17], [18].
The quadcopter is modeled with a state vector consisting of 12 variables:

$$x = \begin{pmatrix} r & v & p & \omega \end{pmatrix}^\top \in \mathbb{R}^3 \times \mathbb{R}^3 \times \mathrm{SO}(3) \times \mathbb{R}^3$$

where:

- $r \in \mathbb{R}^3$ represents the position of the quadcopter in three-dimensional space;
- $v \in \mathbb{R}^3$ is the linear velocity;
- $p \in SO(3)$ is the orientation via Modified Rodrigues Parameters (MRP);
- $\omega \in \mathbb{R}^3$ is the angular velocity.

The control vector u consists of the angular velocities of the four rotors:

$$u = \begin{pmatrix} w_1 & w_2 & w_3 & w_4 \end{pmatrix}^\top \in \mathbb{R}^4$$

where w_i represents the angular velocity of the rotor i.
The dynamics is:

$$\begin{aligned} \dot{r} &= v \\ \dot{v} &= \frac{1}{m}(mg + QF) \\ \dot{p} &= \frac{1}{4}((1 + p^\top p)I + 2\ \mathrm{skew}(p)^2 + 2\ \mathrm{skew}(p))\omega \\ J\dot{\omega} &= \tau - \omega \times (J\omega) \end{aligned}$$

Note that:

- Q is the Direction Cosine Matrix (DCM) for MRP and $F = \begin{pmatrix} 0 & 0 & \sum_{i=1}^4 F_i \end{pmatrix}^\top$ is the total thrust force in body frame.
- J is the inertia matrix of the quadrotor
- $\tau = \begin{pmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{pmatrix}$ is the total torque due to rotor force, with L distance from quadrotor center to rotor, $F_i = k_f \omega_i$ and $M_i = k_m \omega_i$ lift and moment produced by rotor i.

- skew(p) is a Skew-symmetric matrix obtained from the vector p

The quadcopter is modeled as a sphere (*SphereMRP*) with a radius of 0.25 meters, while the obstacles in the environment are described using 12 geometric primitives:

- cylinders (*CylinderMRP*)
- capsules (*CapsuleMRP*)
- cones (*ConeMRP*)
- spheres (*SphereMRP*)
- polytopes (*PolytopeMRP*)
- padded polygons (*PolygonMRP*)

as well as the rectangular prisms used to model the ceiling and floor.
Again, the trajectory optimization problem takes into account collision constraints, state constraints and control constraints. To avoid collisions with the 12 obstacles, the **DCOL** framework was used, while, to plan an optimal trajectory that allows the quadrotor to reach the desired final configuration, the **ALTRO** framework was used. Thanks to the integration of DCOL, ALTRO ensures, once again, a collision-free trajectory respecting all constraints.
The analysis of the results obtained is supported by the images presented in Fig.11, Fig.12 and Fig.13[8], which illustrate the scene and the state and control trajectory of the quadrotor experiment.
It is evident in Fig. 11 how the vehicle successfully avoids all the obstacles present in the three-dimensional environment, respecting the imposed collision constraints.

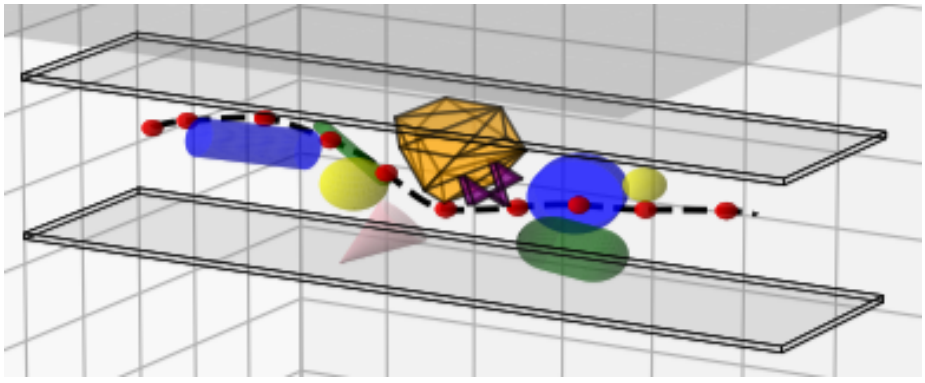

Figure 11: The quadrotor trajectory through the hallway

[8]Note that the quadrotor orientation is shown in euler-angles-form, not with MRP

As visible in Fig.12 the angular velocities of the rotors show significant variations in order to stabilize the quadrotor and initiate movement toward the target. Throughout the trajectory, the commands exhibit oscillatory behavior corresponding to direction changes and critical moments of obstacle avoidance.

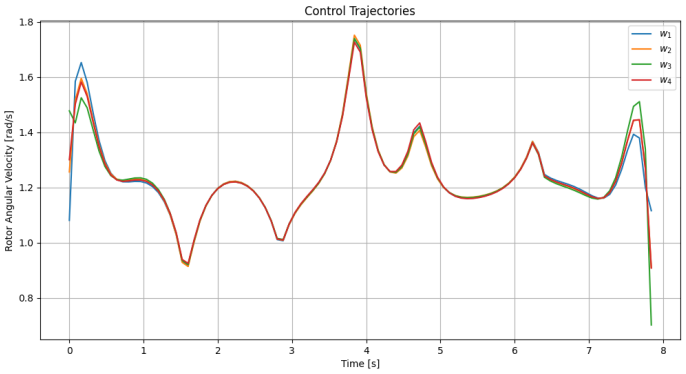

Figure 12: The quadrotor control. Note how the four rotors almost follow the same trajectory

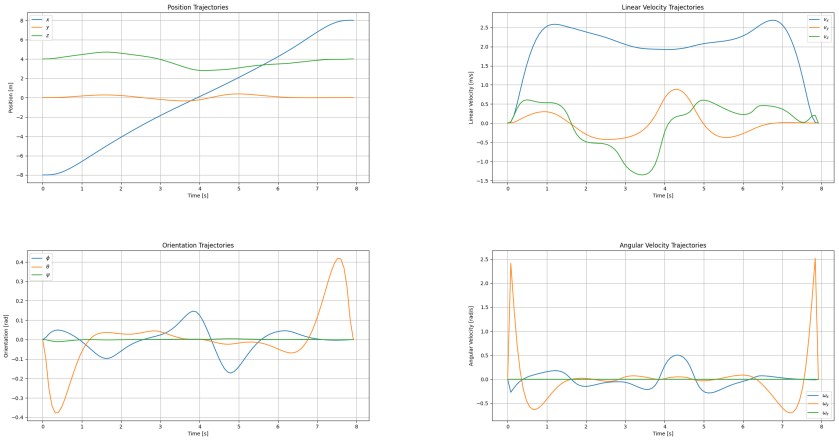

Figure 13: State trajectory of the Quadrotor experiment

The parameters for this experiment are shown in Table 3.

Parameter	**Value**
timesteps (N)	100
dt	0.08
max number of iterations	3000
convergence tolerance (atol)	1×10^{-2}
max linesearch iterations	20
initial regulation	1×10^{-6}
min value for regulation	1×10^{-6}
max value for regulation	1×10^{2}
ρ	1
ϕ	10.0
constraint violation tolerance (convio)	1×10^{-4}
initial state (x_0)	$[-8, 0, 4, 0, 0, 0.0, 0, 0, 0, 0, 0, 0]$
final state (x_g)	$[8, 0, 4, 0, 0, 0.0, 0, 0, 0, 0, 0, 0]$
$u_{\min}$	-2000 (element-wise)
$u_{\max}$	2000 (element-wise)

Table 3: Parameters for the quadrotor experiment.

The algorithm converged in 59 iterations of ALTRO. The time of execution for this experiment are available in Sect 6. Thanks to the combination of the ALTRO and DCOL frameworks, the quadrotor problem has also been successfully addressed and solved, ensuring an optimal trajectory that meets all imposed constraints and allows the quadrotor to safely navigate through the complex environment without collisions.

5.4 Cone Through an Opening

In the *Cone Through Wall* problem, a three-dimensional rigid cone must pass through a square opening in a wall consisting of four rectangular prisms forming a rectangular hole and reach a specified final configuration without collision. The cone has a height of 2 meters and an opening angle of 22 degrees. Obstacles are represented by three-dimensional rectangular prisms of various sizes, positioned in such a way as to make the passage complex and constrained.
The main challenge is to calculate a trajectory that allows the cone to pass through the opening without collision with the surrounding obstacles.
The cone model includes 12 state variables:

$$x = \begin{pmatrix} r & v & p & \omega \end{pmatrix}^T \in \mathbb{R}^3 \times \mathbb{R}^3 \times \mathrm{SO}(3) \times \mathbb{R}^3$$

where:

- $r \in \mathbb{R}^3$ represents the position of the center of mass of the cone;
- $v \in \mathbb{R}^3$ is the linear velocity;
- $p \in SO(3)$ denotes the orientation of the cone using the Modified Rodrigues Parameters;
- $\omega \in \mathbb{R}^3$ is the angular velocity.

The six control variables are:

$$u = \begin{pmatrix} f & \tau \end{pmatrix}^T \in \mathbb{R}^3 \times \mathbb{R}^3$$

where $f \in \mathbb{R}^3$ represents the force applied along the three dimensions, and $\tau \in \mathbb{R}^3$ is the torque applied to the cone.
The dynamics is the following:

$$\begin{aligned} \dot{r} &= v \\ \dot{v} &= \frac{f}{m} \\ \dot{p} &= \frac{1}{4}((1 + p^\top p)I + 2\ \mathrm{skew(p)}^2 + 2\ \mathrm{skew(p)})\omega \\ J\dot{\omega} &= \tau - \omega \times (J\omega) \end{aligned}$$

As for the construction of the primitives, the cone is modeled as a primitive *ConeMRP*, characterized by a height of *2 meters* and an angular opening of *22 degrees*. Instead, the obstacles representing the wall are modeled as three-dimensional rectangular prisms, each defined by varying length, width and height.
The system considers state and control constraints to ensure that the cone does not exceed the imposed physical limits.
To address the problem effectively, the framework **DCOL** was used to handle collision constraints with obstacles, while the framework **ALTRO** allowed

planning an optimal trajectory to the desired final configuration. The synergy between DCOL and ALTRO ensured a solution that complied with all imposed constraints and optimized the cost function, allowing the cone to complete its passage through the aperture without collision.
The results obtained demonstrate the system's ability to deal with the problem effectively. As illustrated in Fig.14, the cone was able to pass through the opening in the wall, respecting all geometrical and dynamics constraints.

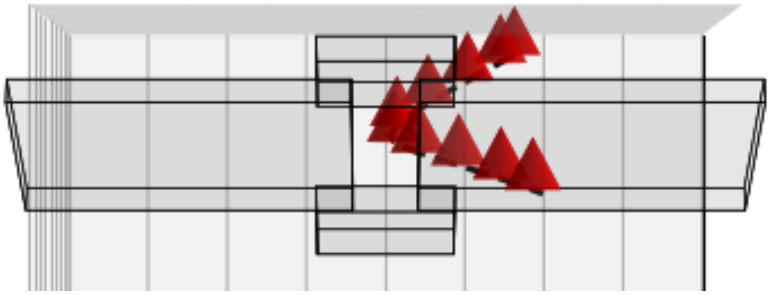

Figure 14: The cone trajectory through the wall

Fig.15[9] and Fig.16 show the evolution of the state and the six controls during the execution of the motion, respectively.

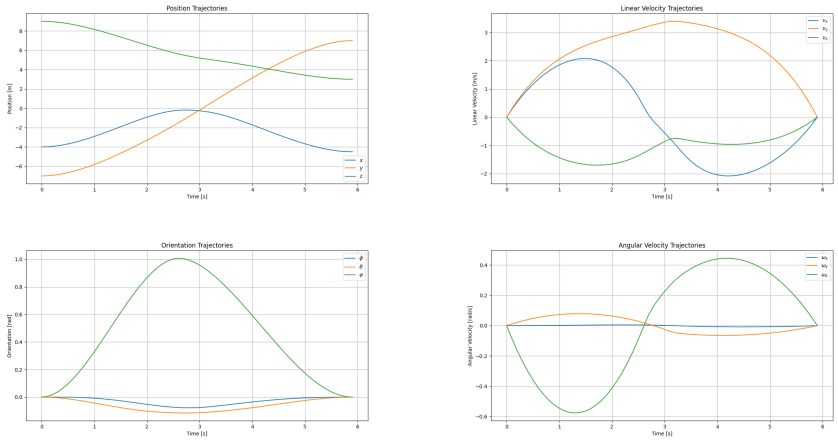

Figure 15: State trajectory of the Cone Through Wall experiment

The parameters for the experiment are shown in 4.

[9]Note that the orientation is shown in Euler-angles-form for visual intuition.

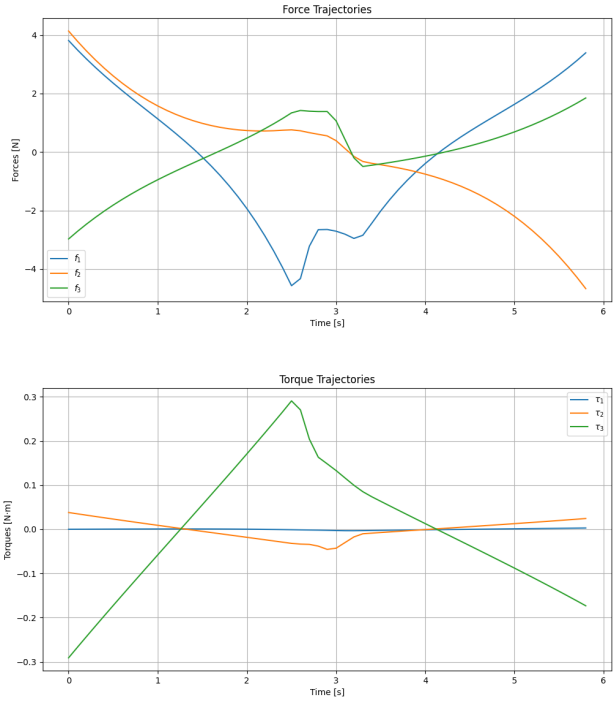

Figure 16: Cone control trajectories.

The algorithm converged in 36 iterations of ALTRO. The time of execution for this experiment are available in Sect 6.

Parameter	**Value**
timesteps (N)	60
dt	0.1
max number of iterations	3000
convergence tolerance (atol)	1×10^{-1}
max linesearch iterations	20
initial regulation	1×10^{-6}
min value for regulation	1×10^{-6}
max value for regulation	1×10^{2}
ρ	1
ϕ	10.0
constraint violation tolerance (convio)	1×10^{-4}
initial state (x_0)	$[-4, -7, 9, 0.0, 0.0, 0.0, 0, 0, 0, 0, 0, 0]$
final state (x_g)	$[-4.5, 7, 3, 0, 0, 0.0, 0.0, 0.0, 0.0, 0, 0, 0]$
$u_{\min}$	-20 (element-wise)
$u_{\max}$	20 (element-wise)

Table 4: Parameters for the quadrotor experiment.

6 Future implementations

This chapter proposes a series of improvements to increase the performance, scalability, and applicability of the framework. First, we address the need to optimize the current Python implementation, which is characterized by a high and unsatisfactory execution time. Next, we explore the integration of methods for the convex decomposition of complex shapes, with the goal of efficiently handling non-convex geometries of any kind. We believe that these optimizations can significantly expand the framework's potential applications, ranging from real-time robotics to advanced simulations in diverse contexts.

6.1 Implementation Speed

One key area for improvement is the speed of the current Python implementation. While Python offers flexibility and ease of prototyping, it often lags in computational efficiency compared to languages like Julia. Benchmarking against existing Julia implementations of similar algorithms has shown significantly faster execution times. A comparison Python/Julia between the three experiments presented previously is shown in Table 5.

Experiment	**Python Time (s)**	**Julia Time (s)**
Piano Mover	104	0.445
Quadrotor	2000 ($\approx$ 30 min)	6.98
Cone Through Wall	300	0.866

Table 5: ALTRO function execution time comparison. Python cProfile and Julia BenchmarkTools were used for the profiling.

The overall code is slow, having no bottleneck or libraries that are responsible for the slowness of the implementation. This is demonstrated by the profiling of our piano mover implementation, in Figs. 17 and 18. To address this problem, a potential future implementation could involve using JAX or other a high-performance library for numerical computing.

6.2 Convex Decomposition

Moreover, methods for convex decomposition of complex shapes can be studied. In this way, the framework can be advanced handling objects with arbitrary, non-convex geometries.

Many objects in real-world robotics and simulations are non-convex, making collision detection more challenging. Convex decomposition involves breaking a complex shape into smaller convex parts that are more manageable for optimization and collision detection algorithms. Decomposition must be computationally efficient to remain viable for real-time applications. Algorithms

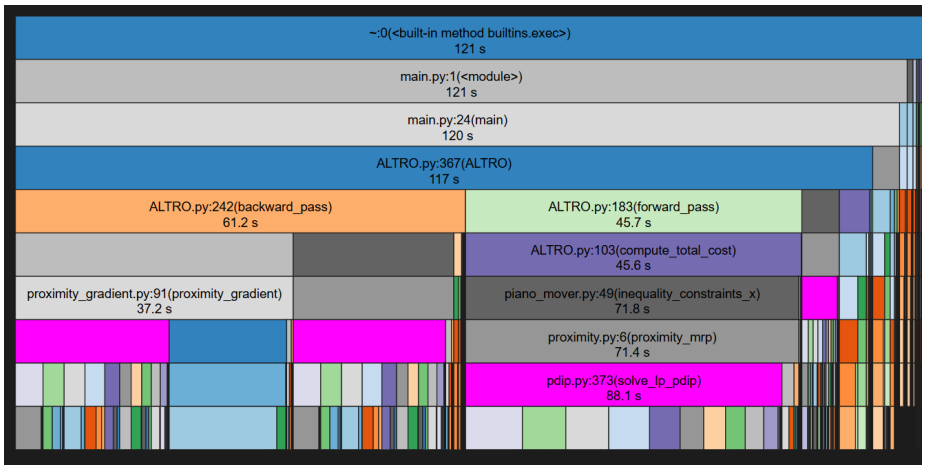

Figure 17: SnakeViz visualization of the time profiling of the piano mover experiment: one of the function that it seems to take most of the time is PDIP, as we would expect.

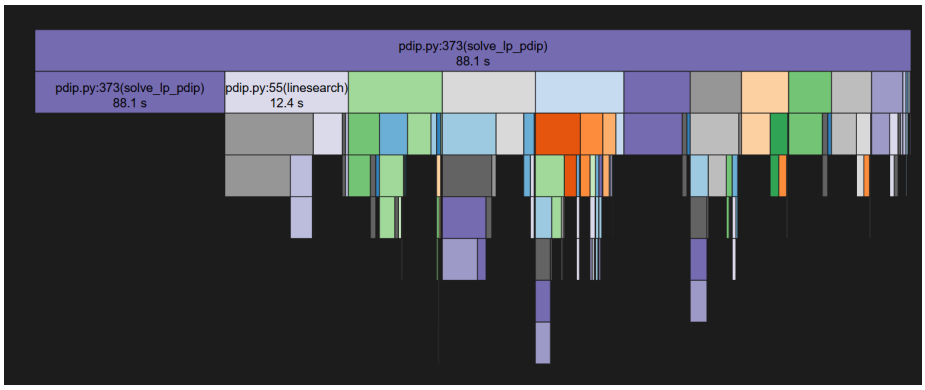

Figure 18: Close up of the PDIP profiling. As you can see, there is no single instruction or function that is considerable a bottleneck, since the cost, in terms of time, is uniform.

like voxel-based decomposition, convex hull generation, or Minkowski sum techniques to generate convex parts can be implemented, machine learning can be leveraged to predict optimal decompositions for commonly encountered shapes. Incorporating convex decomposition into DCOL would enable tackling problems in real-world settings, including environments with intricate geometries like warehouses, urban landscapes, or biological simulations, vastly broadening the scope of its applications.

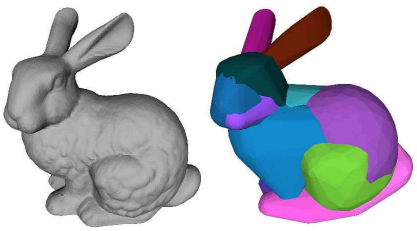

Figure 19: An example of convex decomposition, performed on the 3D model of a bunny

7 Conclusion

In this work, we addressed the problem of collision detection in the context of trajectory planning and optimization for autonomous systems. The proposed methodology, Differentiable Collision Detection (DCOL), represents a significant advance over traditional approaches due to its differentiable formulation and direct integration with gradient-based optimization methods.

To address the motion planning problem, DCOL was integrated into the ALTRO optimization framework, allowing direct handling of collision constraints within the trajectory optimization process. This approach enabled optimal, collision-free trajectories to be obtained efficiently.

The analyzed practical applications, including the Piano Mover problem, the navigation of a quadrotor in a complex environment and the passage of a cone through an opening, demonstrate the feasibility and flexibility of the proposed approach, although very slow compared to the Julia implementation. In each case, the developed framework allowed to generate optimal trajectories respecting the collision constraints.

There is room for improvement in the work done, including optimizing the execution time and extending the framework to handle nonconvex obstacles.
Our open-source Python implementation of DCOL is available at `https://github.com/DIAG-Robotics-Lab/AMR24-FP6-DCOL`.

A Appendix:

Cone Product

The cone product is a specialized operation for vectors in a composite cone structure. For the orthant, it is an elementwise product, while for SOC parts, it follows the algebraic rules specific to second-order cones. For instance, for a composite cone structure, the cone product $\lambda \circ \lambda$ is defined as:

- Orthant Part:
$$(\lambda \circ \lambda)_{\text{ort}} = \lambda_{\text{ort}} \cdot \lambda_{\text{ort}}$$
- SOC Part:
$$\lambda \circ \lambda = \begin{bmatrix} \lambda_0^2 - \|\lambda_{\mathbf{v}}\|^2 \\ 2\lambda_0 \lambda_{\mathbf{v}} \end{bmatrix}$$

Then these two parts are concatenated.

A.1 Inverse Cone Product

For the Orthant Part, the operation here is simple element-wise division:

$$(\lambda^{-1} \circ v)_i = \frac{v_i}{\lambda_i}, \quad \forall i \in \text{orthant indices}.$$

For each SOC, the inverse cone product operation is more complex. It considers the SOC's quadratic structure:

$$v = \lambda^{-1} \circ (-\lambda_\lambda) = \frac{1}{\rho} \begin{bmatrix} \lambda_0(-\lambda_{\lambda,0}) - \langle \lambda_1, -\lambda_{\lambda,1} \rangle \\ \left(\frac{\langle \lambda_1, -\lambda_{\lambda,1} \rangle}{\lambda_0} + \lambda_{\lambda,0} \right) \lambda_1 + \frac{\rho}{\lambda_0} \cdot (-\lambda_{\lambda,1}) \end{bmatrix},$$

where:

- $\rho = \lambda_0^2 - \|\lambda_1\|^2$
- $\nu = \langle \lambda_1, -\lambda_{\lambda,1} \rangle$

Then these two parts are concatenated.

B Appendix:

PDIP Linesearch

The linesearch function computes the maximum allowable step size $\alpha \in [0, 1]$ such that the updated state remains within the feasible region of the composite cone (orthant constraints and second-order cones). We took the minimum between the α_{ort} obtained with orthant linesearch, and α_{soc} obtained with SOC linesearch. The overall step size for each part (orthant and SOCs) is the minimum α across all variables within that part. In the following, the state vector is denoted with y.

B.1 Orthant linesearch

This computes the step size α such that the condition $y_{ort} + \alpha \Delta y_{ort} > 0$ must hold. For the i-th component of y, if $\Delta y_{ort,i} < 0$, the maximum α is

$$-\frac{y_{ort,i}}{\Delta y_{ort,i}}$$

so we took the minimum α within the components of y that respects this condition

B.2 SOC linesearch

This computes the step size α for a second-order cone to ensure that the updated state vector remains feasible within the cone:

$$y_0 + \alpha \Delta y_0 \geq \|\mathbf{y}_v + \alpha \Delta \mathbf{y}_v\|_2$$

Let's breakdown the computation:

1. Quadratic Cone Metric (ν):

$$\nu = \max\left(y_0^2 - \|\mathbf{y}_v\|_2^2, \epsilon\right)$$

   where *epsilon* is a safeguard to prevent division by zero.

2. Dot Product Term (ζ):

$$\zeta = y_0 \cdot \Delta y_0 - \mathbf{y}_v^\top \cdot \Delta \mathbf{y}_v$$

3. Construct Feasibility Direction (ρ): Split $\rho = [\rho_0, \rho_v]$ into scalar and vector components:

$$\rho_0 = \frac{\zeta}{\nu}$$

$$\rho_v = \frac{\Delta \mathbf{y}_v}{\sqrt{\nu}} - \frac{\left(\frac{\zeta}{\sqrt{\nu}} + \Delta y_0\right)}{\left(\frac{y_0}{\sqrt{\nu}} + 1\right)} \cdot \frac{\mathbf{y}_v}{\nu}$$

4. Determine
$$\alpha = \begin{cases} \min\left(1, \frac{1}{\|\rho_v\|_2 - \rho_0}\right) & \text{if } \|\rho_v\|_2 > \rho_0 \\ 1 & \text{otherwise} \end{cases}$$

C Appendix:

Nesterov-Todd Scaling

When $\mathcal{K}$ involves second-order cones (SOCs), NT Scaling matrices are used to handle the SOC structure. Essentially, NT scaling maps each SOC into an equivalent space that resembles the nonnegative orthant. Below, for reference, is the standard Nesterov–Todd (NT) scaling formula for a single second-order cone. Suppose our SOC in $\mathbb{R}^{n+1}$ is

$$\left\{ (x_0, x_1) \in \mathbb{R} \times \mathbb{R}^n \;\middle|\; x_0 \geq \|x_1\| \right\} \tag{53}$$

The Nesterov–Todd scaling matrix W is constructed as follows:

1. Normalize both s and z with respect to the SOC quadratic form:
$$\bar{s} = \frac{s}{\sqrt{s_0^2 - \|s_1\|^2}}, \quad \bar{z} = \frac{z}{\sqrt{z_0^2 - \|z_1\|^2}} \tag{54}$$

2. Compute:
$$\gamma = \sqrt{\frac{1 + \langle \bar{s}, \bar{z} \rangle}{2}} \quad \text{where} \quad \langle \bar{s}, \bar{z} \rangle = \bar{s}_0 \, \bar{z}_0 + \bar{s}_1^\top \, \bar{z}_1 \tag{55}$$

   If $\bar{s}$ and $\bar{z}$ are very aligned, γ is closer to 1. Intuitively, γ controls how we combine the primal and dual directions in the normalized cone space so that the resulting vector remains well-centered and meaningful for the interior-point step.

3. Form the vector $\bar{w}$:
$$\bar{w} = \frac{1}{2\,\gamma}\Big(\bar{s} + \big(\bar{z}_0, \; -\bar{z}_1\big)\Big). \tag{56}$$

   In coordinates, if $\bar{s} = (\bar{s}_0, \, \bar{s}_1)$ and $\bar{z} = (\bar{z}_0, \, \bar{z}_1)$, then
$$\bar{w}_0 = \frac{1}{2\,\gamma}\Big(\bar{s}_0 + \bar{z}_0\Big), \quad \bar{w}_1 = \frac{1}{2\,\gamma}\Big(\bar{s}_1 - \bar{z}_1\Big). \tag{57}$$

   $\bar{w}$ is basically the blend of normalized primal and dual vectors used to built the NT scaling matrix W.

4. Construct the scaling matrix $\bar{W}$ using $\bar{w}$:

$$\bar{W} = \begin{pmatrix} \bar{w}_0 & \bar{w}_1^\top \\ \bar{w}_1 & I_n + b\,\bar{w}_1\,\bar{w}_1^\top \end{pmatrix}, \quad \text{where} \quad b = \frac{1}{\bar{w}_0 + 1}, \tag{58}$$

5. Scale $\bar{W}$ by

$$\eta = \left(\frac{s_0^2 - \|s_1\|^2}{z_0^2 - \|z_1\|^2} \right)^{1/4}. \tag{59}$$

So the final vector is

$$W \;=\; \eta\,\bar{W}. \tag{60}$$

W ensures that the PDIP steps adapt to the geometry of the second-order cone. This helps the solver converge more reliably and efficiently when dealing with SOC constraints.

In our implementation, we computed the NT scaling for a composite cone consisting of an orthant and two SOCs, so we need an additional factor:

$$W_{ort} = \sqrt{\frac{s_{ort}}{z_{ort}}} \quad \text{(elementwise)} \tag{61}$$

Moreover, we perform Cholesky factorizations of the two SOC scaling matrices, since NTScalings should be SPD in the interior of the cone and Cholesky is then a natural choice.

D Appendix:

Cholesky Decomposition

If A is a $n \times n$ SPD matrix, then:

$$A = LL^\top \tag{62}$$

is it's cholesky factorization, with L lower triangular with all entries on the diagonal nonnegative. This factorization enables fast solves, since if we want to solve $Ax = b$, we can do:

$$Ly = b \quad L^\top x = y \tag{63}$$

Avoiding computing A^{-1} explictly, that is more expensive and numerically less stable.

References

[1] K. Tracy, T. A. Howell, and Z. Manchester, "Differentiable collision detection for a set of convex primitives," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3663–3670, IEEE, 2023.

[2] T. A. Howell, B. E. Jackson, and Z. Manchester, "Altro: A fast solver for constrained trajectory optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7674–7679, IEEE, 2019.

[3] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[4] S. Cameron, "Enhancing gjk: Computing minimum and penetration distances between convex polyhedra," in *Proceedings of international conference on robotics and automation*, vol. 4, pp. 3112–3117, IEEE, 1997.

[5] G. Snethen., "Xenocollide: Complex collision made simple," 2008.

[6] J. Newth, "Minkowski portal refinement and speculative contacts in box2d," 2013.

[7] G. Van Den Bergen, "Proximity queries and penetration depth computation on 3d game objects," in *Game developers conference*, vol. 170, p. 209, 2001.

[8] L. Montaut, Q. Le Lidec, A. Bambade, V. Petrik, J. Sivic, and J. Carpentier, "Differentiable collision detection: a randomized smoothing approach," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3240–3246, IEEE, 2023.

[9] S. Boyd and L. Vandenberghe, "Convex optimization," *Cambridge University Press*, 2004.

[10] L. Vandenberghe, "The cvxopt linear and quadratic cone program solvers," p. 30.

[11] E. D. Andersen, C. Roos, and T. Terlaky, "On implementing a primal-dual interior-point method for conic quadratic optimization," *Mathematical Programming*, vol. 95, pp. 249–277, 2003.

[12] Y. E. Nesterov and M. J. Todd, "Primal-dual interior-point methods for self-scaled cones," *SIAM Journal on optimization*, vol. 8, no. 2, pp. 324–364, 1998.

[13] B. Amos and J. Z. Kolter, "Optnet: Differen tiable optimization as a layer in neural networks," *arXiv:1703.00443 [cs, math, stat]*, Oct. 2019.

[14] E. C. A. Domahidi and S. Boyd, "Ecos: An socp solver for embedded systems," in 2013 european control conference (ecc)," *Zurich: IEEE*, Jul. 2013.

[15] J. T. Schwartz and M. Sharir, "On the "piano movers" problem 1. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers," *Communications on Pure and Applied Mathematics*, vol. 36, pp. 345–398, May 1983.

[16] D. Wilson, J. H. Davenport, M. England, and R. Bradford, "A "piano movers" problem reformulated," in *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 53–60, Sept. 2013.

[17] D.Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," *2011 IEEE International Conference on Robotics and Automation (ICRA)*, May 2011, pp. 2520–2525.

[18] K. S. M. S. B. Jackson, T. Howell and Z. Manchester, "Scalable cooperative transport of cable-suspended loads with uavs using distributed trajectory optimization," *International Conference on Robotics and Automation, Paris, France*, Jun. 2020.